KeyAuth

Bringing Public-key Authentication to the Masses

Travis Z. Suel
keyauthsoft@gmail.com

Abstract

Passwords are a fragile, inadequate, and insecure tool for authenticating users, and are especially fraught with problems when used to secure access to network resources and services. In many cases, passwords provide a false sense of security. Creating passwords which are both secure (i.e., hard for attackers to guess) and easy for humans to remember is, at best, a paradoxical task because these two criteria are diametrically opposed. Fortunately, a far more secure and user-friendly alternative is available. Public-key cryptography provides a means of both identifying and authenticating users without the need for passwords. KeyAuth is a generic and universal implementation of public-key authentication aimed at supplanting password-based authentication and significantly improving the security of network accessible resources by enhancing the usability of frequently used authentication mechanisms. KeyAuth is an application-, language-, operating system-, and protocol-independent public-key authentication service.

The Problem

Authentication on the web is *completely broken.*

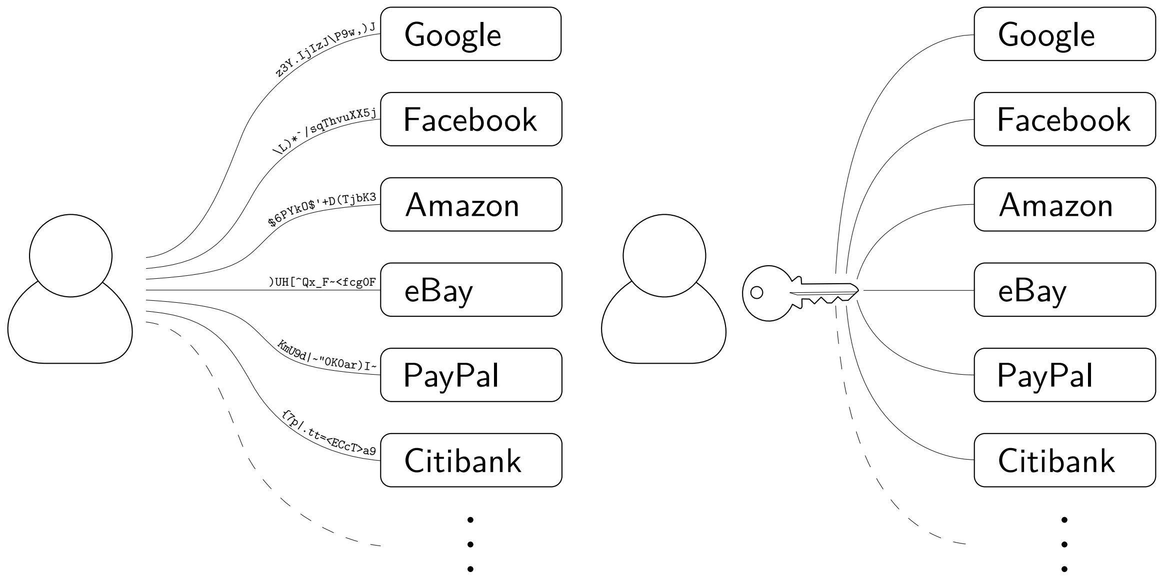
- Passwords are tedious to create.
- Passwords are tedious to use.
- Passwords are tedious to manage.
- Passwords unduly burden end-users.
- Password databases are stolen regularly.
- Authentication is constantly reimplemented.
- All password amelioratives fall short or fail.
- Passwords ultimately degrade security.

One thing is strikingly clear: *Passwords Are Not Working.*

The Solution

- Fixing authentication on the web requires freeing users from passwords. How?
- Replace passwords with public-key authentication.
- SSL supports public-key authentication of users, but nobody uses this facility. Why?
- Public-key authentication of users with SSL is not practical or usable.
- We need a new system designed from the ground up for authenticating users.
- This system must not burden the user with credential management.

KeyAuth **is that new public-key authentication system built specifically for the end-user.**



(a) Password management quickly becomes cumbersome for humans.

(b) KeyAuth allows a user to manage only one set of credentials.

Figure 1: Password authentication (left) vs. KeyAuth (right).

A Fundamentally Different Paradigm

KeyAuth is an authentication system which employs *public-key authentication* (PKA), an authentication paradigm fundamentally different from password-based mechanisms, to verify the identity of users (see Fig. 1). KeyAuth solves the problem of credential reuse allowing end-users to safely authenticate to any number of services with a single set of credentials. Requiring only one set of credentials significantly eases the burden of management. Users are no longer required to memorize long, awkward, and random character strings solving the problems of password management and usability. In addition, KeyAuth effectively eliminates the error-prone generation of passwords. Key pair generation is an automated process which a user has to do only once. Further, KeyAuth, unlike current PKA implementations, uniquely decouples the authentication process and logic from individual applications and operating system facilities. Instead KeyAuth provides public-key authentication as a lightweight, standalone daemon which elides the hardware requirement (e.g., smart cards) of current PKA systems eliminating significant barriers to the adoption of PKA by a broader audience. Providing a standalone service enables software developers to properly factor authentication logic out of their applications allowing them to devote more time and energy towards achieving design goals. Therefore, as the security of KeyAuth improves, so does the security of client applications. Finally, users and organizations do not have to rely on third-parties to perform authentication on their behalf as they would with single sign-on services (e.g., those provided by Google and Facebook, see below). Thus, KeyAuth is a complete replacement for password-based authentication that improves security and usability and can be deployed in environments where smart cards are impractical. KeyAuth

- uses RSA public-key cryptography,
- uses OpenSSL to provide all cryptographic functionality,
- requires no specialized hardware,
- is application-, language-, operating system-, and protocol-independent,
- employs a simple and lightweight JSON protocol, and
- supports non-persistent connections (i.e., authentication over HTTPS).

Table 1: KeyAuth represents a fundamentally different paradigm to authentication with many benefits.

	Password policies	Password lockers	Single sign-on	KeyAuth
Improves usability	—	✓	✓	✓
Simplifies management	—	✓	✓	✓
Obviates generation	—	—	✓	✓
Secures reuse	—	—	partial	✓
No third-party dependencies	✓	—	—	✓
Simplifies implementation	—	—	—	✓
Factors out authentication logic	—	—	—	✓
Fundamentally different paradigm	—	—	—	✓

Next we discuss the principles of public-key authentication and then show that passwords are fundamentally flawed despite the many attempts to fix them. Table 1 illustrates the many advantages of KeyAuth over passwords and the various password amelioratives such as password policies, password safes, and locker services as well as single sign-on (all discussed below).

Public-key Authentication

Public-key authentication utilizes *public-key cryptography* (also referred to as asymmetric cryptography) to identify and authenticate users, rendering passwords obsolete. In public-key cryptosystems, an end-user generates a *key pair*, using an easily automated process, composed of a *public-* and a *private-key*. Public-keys are used to transform information into an unreadable, encrypted form, but they are incapable of reversing that transformation. Only private-keys can decrypt the results. Furthermore, private-keys cannot be derived from public-keys. This property of public-key cryptography allows for more robust and more secure authentication because PKA does not depend on users memorizing passwords. The criteria of usability and security do not compete in PKA systems as in password-based schemes.

In addition, the nature of public-key cryptosystems does not require authentication systems to store sensitive information (e.g., passwords) which could be used to gain unauthorized access. The only information necessary to verify the identity of an end-user is the public-key of that user. Should an attacker successfully obtain a public-key through an unpatched security vulnerability, they would be unable to use that public-key to gain further unauthorized access. Since a public-key can only be used to verify an authorized user's identity, it cannot be used to pose as an authorized user. PKA systems require neither the transmission of a user's private-key, nor its storage in a central location meaning that reusing a key pair, unlike reusing a password, is not fundamentally insecure. Quite the contrary, key reuse is a fully supported use case of properly designed public-key cryptosystems. Potential attackers are also prevented from obtaining valid recoverable credentials from a single intrusion. These aspects of public-key authentication make KeyAuth far more secure and fault-tolerant than passwords, even in the presence of inevitable bugs, security vulnerabilities, and successful breaches.

Public-key authentication is a time-tested and robust approach to user authentication. However, all popular implementations are directly tied to specific applications, operating systems, protocols, or specialized hardware. Transport Layer Security (TLS) and Secure Sockets Layer (SSL) provide PKA facilities. However, this functionality is an extension of identifying web servers and imposes restrictions which, while useful for identifying network nodes, render it impractical for identifying people. When using TLS/SSL to authenticate users, each user's public-key must have an associated certificate provided by a certificate authority. Also, users have to install their key pairs in their browsers which associates public-keys with browser installations, not users. Moreover, this approach limits such authentication to web applications served over HTTPS. Traditional OS-level PKA requires every valid user to have an operating system account on the computer performing authentication and is intended for use with specialized hardware (e.g., smart card readers). Card readers are crucial components for local authentication in secured environments such as government offices, but they serve as an impediment to mainstream adoption as typical end-users do not have access to such

hardware for personal use. Unlike current PKA implementations, KeyAuth disentangles the authentication logic from specific applications, operating systems, and protocols making public-key authentication a practical replacement for passwords.

Passwords are the Problem

For a password to be secure it must be difficult for both a human and a computer to guess, yet easy to remember. This security is typically achieved by creating passwords as strings of random characters which include letters of both cases, numbers, and various other symbols, including punctuation and brackets [3]. As the length of these random amalgams of characters increase, so does the difficulty of guessing them. However, as password complexity and length increases, passwords become more challenging for humans to memorize and type; there is a steep drop in usability because passwords are not compatible with the mechanisms of human memory. Human memory employs a strategy called *chunking* in which an individual improves recall by dividing information into chunks, such as words or syllables, which are perceived to have meaning [1]. Unfortunately, strings of random characters do not easily decompose into retainable chunks. Thus, choosing between password security and usability is a zero-sum exercise. Passwords are an inherently ineffective credential mechanism because they rely directly on human memory [6, 7].

Implementing and deploying password-based authentication is a common source of security vulnerabilities. Any such system must store user passwords in some form. Because storing passwords in plaintext is a security vulnerability, passwords are often stored using a one-way hash algorithm (e.g., MD5 or SHA1). However, this is often done improperly without a *salt** or with a salt common to all users. Regardless of how securely passwords are stored, maintaining them in a central location conveniently provides an attacker with a single point of access. If an attacker discovers a means of unauthorized access to this highly sensitive information (possibly by exploiting a SQL injection vulnerability), they can mount an offline dictionary attack and likely recover passwords from a subset of users and gain further unauthorized access. Even software which properly salts passwords before hashing is vulnerable because of password reuse. An attacker who gains unauthorized access to an insecure but non-critical system, perhaps a public-facing content management system, can reasonably expect some recovered passwords to be valid with other more secure and high-value systems, such as source code repositories or e-mail servers [2]. This attack vector can be effective both within and across organizational boundaries.

Attack vectors arise from password reuse because most organizations and software implement their own authentication logic. Continually re-implementing authentication logic (within and between organizations) not only wastes productivity but also increases the number of systems with potentially exploitable security flaws which expands attack surfaces. An attacker can exploit such flaws in a vulnerable system to recover passwords and subsequently access otherwise inaccessible secure systems. There are no compelling reasons why the software industry should take such a haphazard approach to implementing authentication logic. Rather, this effort should be devoted to achieving the unique design goals of individual applications. Moreover, the plurality of disjoint implementations means as bugs are fixed in one application, similar or identical bugs may go unnoticed and unpatched in others. As a result, passwords and the common approaches taken to implementing them are irredeemably flawed approaches to the problem of authenticating users. In summary, password vulnerabilities are the result of many factors:

$$\textit{password vulnerabilities} = \left\{ \begin{array}{c} \textit{password} \\ \textit{reuse} \end{array} \times \begin{array}{c} \textit{centralized} \\ \textit{datastore} \end{array} \times \begin{array}{c} \textit{logic} \\ \textit{errors} \end{array} \times \begin{array}{c} \textit{abundance of} \\ \textit{implementations} \end{array} \right\}$$

Passwords Cannot Be Fixed

Password Policies

Many organizations attempt to improve the security of passwords and prevent reuse by imposing *password policies* on users which attempt to promote secure behavior. However, password policies are often examples of

*A string of random characters appended to a password before hashing to defend against dictionary attacks.

anti-patterns[†] and suffer from serious unforeseen consequences [3]. These policies seek to improve password security by requiring users to create highly complex passwords of a secure length, and, in many cases, requiring users to change passwords on a regular basis. These policies, while well-intentioned, are frequently overzealous in their complexity and, thus, unrealistic for users to abide. Ultimately, password policies compel users to store passwords in plaintext, often by writing them down, and compromise their security. Users often have no other choice as they cannot memorize a constant stream of new, complex passwords. Password policies also increase IT support costs as more users are required to call for help when they invariably forget their new, secure password [3].

A single password policy is composed of multiple requirements such as minimum length, required character types, and password expiration. Compounding this problem further, many organizations are subject, internally, to multiple password policies. This makes creating new passwords difficult because these various policies are often incompatible and failing to post clear and explicit password requirements is confoundingly common. One password policy may prohibit a specific set of characters while another policy may ban all non-alphanumeric characters. Still another policy may impose an arbitrary maximum length. As an example of the complexities such policies introduce, performing a Google search for "`amazon password policy`" yields no helpful results from Amazon. When changing an Amazon password, if a user enters an invalid password (perhaps containing disallowed characters), Amazon silently rejects the password leading the user to believe it has been accepted. Password policies struggle to treat the symptoms of passwords while further complicating their use and ignoring the root of the problem.

Password Safes

To cope with the ever-increasing number and complexity of passwords, some users choose to employ *password safes* or *lockers* to manage their myriad credentials. A password safe can either manage a locally stored, usually encrypted, database, or it can manage a database stored in a remotely accessible location. Password safes which manage a local database (e.g., KeePass) help to alleviate the user of remembering long complex passwords without resorting to plaintext storage. However, if a user owns multiple devices (e.g., a laptop, a tablet, and a smartphone), they must synchronize the database on each device every time they add, update, or delete a password entry. Password locker services (e.g., LastPass) relieve the user of password database management by storing the database "in the cloud" making it easily accessible to multiple devices.

Password safes, while mitigating poor usability and cumbersome management, do not fix passwords. Most passwords lockers are still secured using a password, and locker services exacerbate the aspect of centralized storage since passwords to multiple services are collected from many users. Safes and lockers also do nothing to solve the problem of generation and only partially remedy reuse. Creating a new account or changing the password on an existing account still requires a person to know the password requirements of a particular service. Further, locker services also make people dependent on third-parties to store and maintain their passwords. It is neither unheard of nor unreasonable for industry heavyweights, including Google and Microsoft, to suffer the occasional service outage. Outages of Gmail or Office 365 may be a nuisance or significant interruption, but, if a locker service suffers an outage, their users are effectively locked out of all other services since access to all credentials has been temporarily suspended. Lastly, though importantly, password safes only prevent the fallout of reuse for people who chose to use them. Businesses and organizations are still subject to the security implications of password reuse by those who do not.

Single Sign-on

Single sign-on (SSO) systems only partially mitigate the fundamental flaws of passwords. These systems, ultimately, still attempt to derive their security from passwords, and users are still required to authenticate with SSO services using passwords. If a password or SSO service is comprised, all systems which rely on that SSO service are suddenly susceptible to security breaches. Moreover, using the SSO service of another organization introduces third-party dependencies. Supplementary to local accounts, all users need accounts with the SSO provider, and an organization must forfeit some level of control to that provider. A provider may choose to alter their protocol in a way that is not backwards compatible or even discontinue service entirely. The client organization may have little recourse but to bear the cost of retooling their software or

[†]A solution which is commonly used in software development, but which is either ineffective or counterproductive.

migrating to another, possibly incompatible, service. A client organization may inadvertently run afoul of the business policies of their SSO provider and have their access revoked without prior notice, a scenario in which Grooveshark recently found themselves [4]. Finally, SSO systems introduce a great deal of complexity to the authentication process rendering them difficult to implement properly and securely [5]. Some SSO flaws have even been found to allow attackers unauthorized access without obtaining any passwords. Compounding the intricacies of SSO, many organizations are required to support multiple SSO providers because their users do not share the same provider. While single sign-on clearly minimizes certain undesirable aspects of passwords, it decidedly exacerbates others.

Conclusion and Further Reading

Usability is the key to promoting secure behavior in end-users. Make security an easy and convenient habit and users will adopt secure conduct. Passwords are insecure because they burden users with extraneous and repetitive work which unintentionally fosters bad behavior. Most password amelioratives do solve some of the shortcomings of passwords but ignore or even aggravate others. Some of these purported solutions even introduce new problems. Unfortunately, none of the amelioratives are capable of solving all the problems inherent to password-based authentication (see Table 1). KeyAuth bypasses these problems entirely by obviating the need for passwords with a fundamentally different approach to user authentication and, therefore, relieves people of all the associated tedious and error-prone management and overhead of passwords. The onus of herding the mundane and tiresome details of credential security shifts from humans to software. With KeyAuth, instead of being arduous, frustrating, and time consuming, proper security becomes innate, intrinsic, and natural.

[1] Chunking (psychology). Wikipedia. Available at `http://en.wikipedia.org/wiki/Chunking_(psychology)`.

[2] Bright, P. Anonymous Speaks: The Inside Story of the HBGary Hack. *Ars Technica* (February 2011). Available at `http://arstechnica.com/tech-policy/2011/02/anonymous-speaks-the-inside-story-of-the-hbgary-hack/` (short URL: `http://goo.gl/7EAXu`).

[3] Gallagher, S. When Passwords Attack: The Problem with Aggressive Password Policies. *Ars Technica* (October 2011). Available at `http://arstechnica.com/business/2011/10/when-passwords-attack-the-problem-with-aggressive-password-policies/` (short URL: `http://goo.gl/1fTet`).

[4] Geuss, M. Facebook Deliberately Bans Grooveshark from Its Services. *Ars Technica* (May 2012). Available at `http://arstechnica.com/gadgets/2012/05/facebook-deliberately-bans-grooveshark-from-its-services/` (short URL: `http://goo.gl/EQidm`).

[5] Goodin, D. Flawed Sign-in Services from Google and Facebook Imperil User Accounts. *Ars Technica* (March 2012). Available at `http://arstechnica.com/business/2012/03/flawed-sign-in-services-from-google-and-facebook-imperil-user-acconts/` (short URL: `http://goo.gl/TmtxQ`).

[6] Goodin, D. Passphrases Only Marginally More Secure than Passwords Because of Poor Choices. *Ars Technica* (March 2012). Available at `http://arstechnica.com/business/2012/03/passphrases-only-marginally-more-secure-than-passwords-because-of-poor-choices/` (short URL: `http://goo.gl/6uA1z`).

[7] McMillan, R. The World's First Computer Password? It Was Useless Too. *Wired* (January 2012). Available at `http://www.wired.com/wiredenterprise/2012/01/computer-password/` (short URL: `http://goo.gl/eIYLF`).